Aufgabe 3: Zauberschule

Team-ID: 00889

Team: Felix Haag

Bearbeiter/-innen dieser Aufgabe:
Felix Haag

12. November 2023

Inhaltsverzeichnis

Lösungsidee

Zum Finden des schnellsten Weges von A nach B kann der Lee-Algorithmus (Breitensuche) verwendet werden. Hierbei wird das Labyrinth vom Startpunkt aus, bei jedem Schritt „gleichzeitig“ in alle Richtungen durchlaufen, bis der Endpunkt gefunden wird. Man startet beim Startfeld und besucht jedes Nachbarfeld in alle möglichen Richtungen. Anschließend besucht man von jedem dieser Nachbarfelder wieder alle Nachbarfelder. Dies wird solange wiederholt bis das Endfeld gefunden wurde.

Von einem Feld zum nächsten kann prinzipiell nach oben, rechts, links und unten gegangen, sowie das Stockwerk nach oben oder unten gewechselt werden. Da beim Lee-Algorithmus in alle Richtungen gleichzeitig gegangen wird, das Wechseln des Stockwerkes jedoch 2s länger dauert als das Bewegen innerhalb eines Stockwerkes, müssen hier noch zwei Zwischenstockwerke eingebaut werden, in denen nur weiter nach oben oder unten gewechselt werden kann.

Umsetzung

Vorbereitung

Die Lösungsidee wird in Python implementiert. Zuerst wird die Eingabedatei eingelesen. Es werden drei dreidimensionale Listen für Stockwerk, Reihe und Spalte erstellt. In der ersten Liste wird der Aufbau der Stockwerke gespeichert. Ein freies Feld wird hier durch „True“ gekennzeichnet, eines mit Wand durch „False“. Eine zweite Liste speichert welche Felder beim suchen des Weges bereits

besucht wurden. In der dritten wird die Karte mit den Zeichen der Eingabedatei („#“ und „.“) gespeichert. Diese wird später für die Ausgabe verwendet.

Dazu wird die Eingabedatei Zeichen für Zeichen durchlaufen. Das jeweilige Zeichen wird in der Ausgabe-Liste an seiner Position gespeichert. Ist das Zeichen ein „#“ wird das Feld in der ersten Liste als „False“ markiert. Ist das Zeichen „A“ oder „B“ werden noch die Koordinaten des Start- beziehungsweise Endfeldes gespeichert. Überhalb des oberen Stockwerkes und unterhalb des unteren Stockwerkes wird jeweils noch ein Stockwerk voller Wände (False) eingefügt, um zu verhindern, das vom oberen Stockwerk noch weiter nach oben und vom unteren noch weiter nach unten gewechselt werden kann. Zusätzlich werden zwischen den beiden Stockwerken noch zwei leere Stockwerke gefüllt mit „True“ eingefügt, um die Dauer von 3 Sekunden eines Stockwerkwechsels zu berücksichtigen.

Es wird außerdem noch eine Klasse „Node“ erstellt, die für jeden Wegpunkt bei der Suche des Weges die Koordinaten, die Richtung aus der man zu ihm gelangt ist und eine Referenz zum vorherigen Wegpunkt speichert. In einem Enum sind alle möglichen Richtungen (hoch, rechts, runter, links, Wechsel nach oben, Wechsel nach unten) mit den jeweiligen Änderungen der Koordinaten und Zeichen für die Ausgabe (^, >, v, <, !) gespeichert.

Algorithmus

Es wird eine Funktion definiert, die die Liste für die Karte und für die bereits besuchten Wegpunkte, sowie die Koordinaten des Start- und Endpunktes übergeben bekommt. Für den Algorithmus wird eine Queue verwendet, die alle als nächstes zu betrachtenden Wegpunkte beinhaltet. Zu dieser wird zuerst der Startknoten mit den Koordinaten des Startfeldes hinzugefügt und das Startfeld in der dafür vorgesehen Liste als besucht markiert.

Nun wird die Queue mit einer while-Schleife durchlaufen, solange bis sie leer ist. Bei jedem Durchlauf wird der vorderste Knoten der Queue entnommen. Danach werden alle Richtungen des Direction-Enums mit einer for-Schleife durchlaufen. Befindet sich der entnommene Knoten in einer der beiden Zwischenebenen, wird nur das wechseln des Stockwerkes akzeptiert, alle anderen Richtungen werden übersprungen. Es werden die neuen x-, y- und z-Koordinaten berechnet, die nach Gehen in die jeweilige Richtung entstehen würden. Dafür werden die Werte der Richtung aus dem Enum zu den Koordinaten des entnommenen Knotens aus der Queue addiert. Zuerst wird überprüft, ob die neuen Koordinaten bereits dem gesuchten Ende entsprechen. Wenn ja, gibt die Funktion einen neuen Knoten mit diesen Koordinaten, der gegangenen Richtung und der Referenz zum vorherigen Knoten zurück. Dies ist dann der Fall, wenn der schnellste Weg von A nach B gefunden wurde. Ansonsten wird überprüft ob an den neuen Koordinaten auf der Karte ein freies begehbares Feld und keine Wand ist und das Feld noch nicht besucht wurde. Trifft dies zu, wird das Feld als besucht markiert und ein neuer Knoten mit den neuen Koordinaten, der gegangenen Richtung und Referenz zum vorherigen Knoten erstellt und zur Queue hinzugefügt. Ist die Queue leer, wurde kein Weg gefunden und es wird „None“ zurückgegeben.

Ausgabe

Für die Ausgabe wird die Linked-List des kürzesten Weges, deren Start- bzw. Endknoten von der Suchfunktion zurückgegeben wurde, mit einer while-Schleife durchlaufen. Bei jedem Durchlauf wird 1s zur Variable für die Zeit des Weges hinzugefügt und von jedem Knoten des Weges werden die Koordinaten und die Richtung zu einer Liste hinzugefügt. Diese Liste wird dann von hinten nach vorne, also von Start bis Ende des Weges durchlaufen und für jeden Wegpunkt das Zeichen der Richtung an der jeweiligen Stelle der Liste mit den Zeichen der Eingabe eingesetzt. Zum Schluss wird diese Liste in einen String umgewandelt und zusammen mit der Zeit des Weges in der Konsole ausgegeben und in einer Textdatei gespeichert.

Beispiele

zauberschule0.txt

Zeit des schnellsten Weges: 8 Sekunden

Oben:

```
#############
#.....#.....#
#.###.#.###.#
#...#.#...#.#
###.#.###.#.#
#...#.....#.#
#.#########.#
#.....#.....#
#####.#.###.#
#....A#!..#.#
#.#########.#
#...........#
#############
```

Unten:

```
#############
#.......#...#
#...#.#.#...#
#...#.#.....#
#.###.#.#.###
#.....#.#...#
#####.###...#
#.....#.....#
#.#########.#
#...#>>.....#
#.#.#.#.###.#
#.#...#...#.#
#############
```

zauberschule1.txt

Zeit des schnellsten Weges: 4 Sekunden

Oben:

```
####################
#...#.....#...#.....#
#.#.#.###.#.#.#.###.#
#.#.#...#.#.#...#...#
###.###.#.#.#####.###
#.#.#...#.#B....#...#
#.#.#.###.#^###.#####
#.#...#.#.#^<<#.....#
#.#####.#.#########.#
#..................#
####################
```

Unten:

```
####################
#.......#.....#.....#
#.###.#.#.###.#.###.#
#.....#.#.#.....#.#.#
#######.#.#######.#.#
#.....#.#.....#...#.#
#.###.#.#.###.###.#.#
#.#.#...#.#...#...#.#
#.#.#######.###.###.#
#...........#.......#
####################
```

zauberschule2.txt

Zeit des schnellsten Weges: 14 Sekunden

Oben:

```
################################################
#...#.....#...........#.#...........#.......#
#.#.#.###.#########.#.#.#.#######.#.#.#####.#
#.#.#...#.#.........#>>.#v#.....#.#.#...#...#
###.###.#.#.##############v#.#.###.#.###.#.###
#.#.#...#.#..............>>B#.#...#.#...#.#.#
#.#.#.###.###########.#########.###.#.###.#.#
#.#...#.#.#.........#.#.#.....#.#.....#.#.#.#
#.#####.#.#.#######.#.#.#.###.#.#######.#.#.#
#.....#...#...#.#...#...#.#.#.#.......#.#.#.#
#.#####.#####.#.#.#######.#.#.#######.#.#.#.#
#.....#.......#.#.#.....#.#.#.#...#...#...#.#
#.###.#########.#.###.#.#.#.#.#.#.#.###.###.#
#...#.................#...#.....#...#.......#
################################################
```

Unten:

```
################################################
#...#.......#.....#.....#...#...#.....#.....#
#.#.#.#####.###.#.###.#.#.#.#.#.#.###.###.###
#.#.#.....#.#...#.....#>>.#.#.#...#.#...#...#
###.#.###.#.#.##############.#.#####.###.###.#
#.#.#...#.#.#.#.....#...#.#.#...#.#...#.#...#
#.#.#####.#.#.#.###.#.#.#.#.#.#.#.#.#.#.#.###
#.#...#...#.#.....#.#.#...#.#.#.#...#.#.#...#
#.###.#.###.#.#####.#.###.#.###.#####.#.###.#
#...#.#.#...#...#...#...#.#...#.#.....#.....#
#.###.#.#.#######.#####.#####.#.#.#.#######.#
#...#...#...#...#.....#.......#.#.#.#.....#.#
#.#.#######.#.#.#####.#.#######.#.###.###.#.#
#.#...........#.......#.........#.......#...#
################################################
```

zauberschule3.txt

Zeit des schnellsten Weges: 28 Sekunden

Oben:

```
################################
#...#.....#..........#.......#
#.#.#.###.#.###.#######.###.#.#
#.#.#...#.#.#.#.#.......#...#.#
###.###.#.#.#.#.#.#######.###.#
#.#.#...#.#...#.....#.....#.#.#
#.#.#.###.###########.#####.#.#
#.#...#.#.#.........#...#.....#
#.#####.#.#.#######.###.#.#####
#...#.#...#...#.#...#...#.#...#
#.#.#.#.#####.#.#.###.###.#.#.#
#.#.#.#.......#.#.#...#.#...#.#
#.#.#.#########.#.#.###.#####.#
#.#.......#.#.....#.#.#.....#.#
#.#######.#.#.#####.#.#.###.#.#
#.#.....#...#.#.#...#.#.#.#...#
#.#.###.#####.#.#.#.#.#.#.#####
#.#...#.....#.#.#.#.#.#...#.....#
#.###.#####.#.#.#.#.#.###.#####.#
#...#.#...#...#...#...#.......#
#.#.#.#.##############.#.#######
#.#.#.#...............#.#.....#
#.###.#.#######.#########.###.#
#...#.#.#...#...#.........#...#
###.#.###.#.#.###.#####.###.#.#
#...#.....#.#.....#>>B#.#...#.#
#.#########.#######^#.###.###.#
#..v#>>>>v#.#>>>>>>^#...#.#.#.#
#.#v#^###v#.#^#########.#.#.#.#
#.#>>^..#>>>>^#...........#...#
################################
```

Unten:

```
################################
#.......#.....#...#...#.......#
#.###.#.#.#.#.#.###.#.#.#.#####
#.....#.#.#.#.#.....#.#.#.....#
#######.#.#.#.#.#####.#.#####.#
#.....#.#.#.#.#.#...#...#.....#
#.###.#.###.#.###.#.#.###.#####
#...#.#.#...#.....#.#.#.#.....#
#.#.###.#.#########.#.#.#####.#
#.#.....#.#.......#.#.....#...#
#.#######.#######.#.#.#####.#.#
#...#...#.....#...#.#.#...#.#.#
###.#.#.#.###.#.###.#.#.#.#.#.#
#.#.#.#...#...#.....#.#.#.#.#.#
#.#.#.#####.###.#######.#.#.#.#
#.#.#.....#.#.....#.....#.#.#.#
#.#.#######.#####.###.###.#.#.#
#.#.....#...#...#.....#.#...#.#
#.#####.#.###.#.#######.#####.#
#.#...#.#.#...#.......#.#.....#
#.#.#.#.#.#.#########.#.#.#####
#...#.#.#.#...#...#.#.#.#.....#
#####.#.#.###.#.#.#.#.#.#####.#
#.....#.#.......#.#.#.#...#...#
#.###.#.#########.#.#.#.#.#.###
#...#.#.....#.#.....#.#.#.#.#.#
#.#.#######.#.#.#####.#.#.#.#.#
#.#.......#.#...#.....#.#...#.#
#.###.#####.#####.#####.#####.#
#...#.............#...........#
################################
```

zauberschule4.txt

Zeit des schnellsten Weges: 84 Sekunden

zauberschule5.txt

Zeit des schnellsten Weges: 124 Sekunden

Quellcode

Einlesen der Textdatei

```
def read_file(filename):
    map = [] # Liste zum Speichern des Labyrinths
    discovered = [] # Liste des Labyrinths, die angibt welche Felder schon besucht werden
    char_map = [] # Liste des Labyrinths für Ausgabe
    with open(os.path.dirname(__file__) + "/" + filename, "r") as f:
        dims = f.readline().split(" ")
        height = int(dims[0].replace("\n", "")) # Höhe des Labyrinths
        width = int(dims[1].replace("\n", "")) # Breite des Labyrinths

        input = f.read()
        floors = input.split("\n\n")
        floors = [f for f in floors if f] # Stockwerke des Labyrinths

       # Ein Stockwerk voller Wände, damit vom obersten Stockwerk nicht noch höher gegangen werden kann
        map.append([[False for _ in range(width)] for _ in range(height)])
        discovered.append([[True for _ in range(width)] for _ in range(height)])

        # Durchlaufen der beiden Stockwerke
        for f, floor in enumerate(floors):
            # Stockwerk mit True füllen, True = Freies Feld ohne Wand
            map.append([[True for _ in range(width)] for _ in range(height)])

            # Stockwerk mit False füllen, False = Feld wurde noch nicht besucht
            discovered.append([[False for _ in range(width)] for _ in range(height)])
            char_map.append([]) # Stockwerk zur Ausgabe-Liste hinzufügen
            rows = floor.split('\n') # Stockwerk in Reihen splitten

            # Durchlaufen der Reihen des Stockwerks
            for r, row in enumerate(rows):
                char_map[f].append([]) # Reihe zur Ausgabe-Liste hinzufügen
                # Durchlaufen der Felder der Reihe
                for c, char in enumerate(row):
                    char_map[f][r].append(char) # Zeiche zur Ausgabe-Liste hinzufügen

                    # Falls auf Feld eine Wand steht
                    if char == '#':
                        map[-1][r][c] = False # Feld als Wand auf Karte markieren

                    # Falls das Feld das Startfeld ist
                    if char == "A":
                        start = (c, r, len(map)-1) # Speichern der Startposition
                    # Falls das Feld das Endfeld ist
                    elif char == "B":
                        end = (c, r, len(map)-1) # Speichern der Endposition

            # Falls oberstes Stockwerk
            if f == 0:
                # Einfügen von zwei leeren Stockwerken zwischen den eigentlichen Stockwerken, für 3
                  Sekunden Dauer des Stockwerkwechsels
                map.append([[True for _ in range(width)] for _ in range(height)])
                map.append([[True for _ in range(width)] for _ in range(height)])
                discovered.append([[False for _ in range(width)] for _ in range(height)])
                discovered.append([[False for _ in range(width)] for _ in range(height)])

        # Ein Stockwerk voller Wände, damit vom untersten Stockwerk nicht noch tiefer gegangen werden kann
        map.append([[False for _ in range(width)] for _ in range(height)])
        discovered.append([[True for _ in range(width)] for _ in range(height)])

    return char_map, map, discovered, start, end
```

Node-Klasse und Direction-Enum

```
# Wegpunkt bei der Suche eines Weges
class Node:
    def __init__(self, x, y, z, initial_dir, previous_node):
        self.x = x
        self.y = y
        self.z = z
        self.initial_dir = initial_dir
        self.previous_node = previous_node # Referenze zum vorherigen Wegpunkt

# Alle Richtungen in die von einem zum nächsten Feld gelaufen werden kann
class Direction(Enum):
    UP = (0, -1, 0, "^")
    RIGHT = (1, 0, 0, ">")
    DOWN = (0, 1, 0, "v")
    LEFT = (-1, 0, 0, "<")
    SWITCH_UP = (0, 0, -1, "!") # Wechseln des Stockwerkes nach oben
    SWITCH_DOWN = (0, 0, 1, "!") # Wechseln des Stockwerkes nach unten
```

Finden des schnellsten Weges

```
# Lee Algorithmus
def find_fastest_path(map, discovered, start, end):
    queue = [] # Queue für Wegpunkte, die als nächstes betrachtet werden müssen
    queue.append(Node(start[0], start[1], start[2], None, None)) # Startpunkt des Weges
    discovered[start[2]][start[1]][start[0]] = True # Feld als besucht markieren

    # Solange die Queue nicht leer ist
    while not(len(queue) == 0):
        node = queue.pop(0) # Herausnehmen des vordersten Wegpunktes aus der Queue

        # Durchlaufen aller Richtungen, in die von einem Wegpunkt gegangen werden kann
        for dir in Direction:
            # Falls in Zwischenstockwerk, nur Stockwerk kann gewechselt werden
            if node.z != 1 and node.z != 4 and dir != Direction.SWITCH_DOWN and dir != Direction.SWITCH_UP:
                continue

            newX = node.x + dir.value[0] # Berechnen der neuen x-Koordinate nach gehen in diese Richtung
            newY = node.y + dir.value[1] # Berechnen der neuen y-Koordinate nach gehen in diese Richtung
            newZ = node.z + dir.value[2] # Berechnen der neuen z-Koordinate nach gehen in diese Richtung
                                                                                    (Stockwerk)

            # Falls die neue Position dem gesuchten Ende entspricht
            if newX == end[0] and newY == end[1] and newZ == end[2]:
                # Zurückgeben des letzten Wegpunktes des gefundenen Weges zum Ziel
                return Node(newX, newY, newZ, dir, node)

            # Falls an dieser Position im Labyrinth keine Wand steht und das Feld noch nicht besucht wurde
            if map[newZ][newY][newX] == True and not(discovered[newZ][newY][newX]):
                discovered[newZ][newY][newX] = True # Wegpunkt als besucht markieren
                queue.append(Node(newX, newY, newZ, dir, node)) # Wegpunkt zur Queue hinzufügen

    return None # Kein Weg wurde gefunden
```

Ausgabe

```
def create_output_map(char_map, path, start):
    previous_field = start
    # Durhclaufen des Weges
    while len(path) > 0:
        step = path.pop() # Herausnehmen des letzten Wegpunktes

        # Nur Bewegungen im oberen und unteren Stockwerk müssen berücksichtigt werden
        if step[2] == 1:
            # Speichern des Richtungszeichen an der jeweiligen Stelle der Karte (oberes Stockwerk)
            char_map[0][previous_field[1]][previous_field[0]] = step[3].value[3]
        elif step[2] == 4:
            # Speichern des Richtungszeichen an der jeweiligen Stelle der Karte (unteres Stockwerk)
            char_map[1][previous_field[1]][previous_field[0]] = step[3].value[3]
        previous_field = (step[0], step[1], step[2])

    output_string = ""

    # Liste in String umwandeln
    for floor in char_map:
        for row in floor:
            for col in row:
                output_string += col
            output_string += '\n'
        output_string += "\n"

    return output_string # String zurückgeben

def output(char_map, end_node, filename):
    time = 0
    path = []

    new_node = end_node
    # Durchlaufen des Weges von hinten nach vorne (Linked List)
    while(new_node.previous_node != None):
        time += 1 # Bei jedem Schritt 1s zur Zeit addieren

        # Koordinaten und Richtung zum Weg hinzufügen
        path.append((new_node.x, new_node.y, new_node.z, new_node.initial_dir))
        new_node = new_node.previous_node # Aktuellen Wegpunkt auf vorherigen wegpunkt setzen

    output_map = create_output_map(char_map, path, start) # Weg in Output-Karte eintragen

    print(f"Zeit: {time} Sekunden\n") # Zeit ausgeben
    print(output_map) # Karte mit Lösung ausgeben

    save_output(output_map, time, filename) # Lösung in Textdatei speichern

def save_output(output_map, time, filename):
    with open(os.path.dirname(__file__) + f"/{filename.split('.')[0]}_Loesung.txt", "w") as f:
        f.write(f"Zeit: {time} Sekunden\n")
        f.write(output_map)
```

Ausführung

```
if __name__ == '__main__':
    filename = ""

    while not(filename in os.listdir(os.path.dirname(__file__))):
        filename = input("Dateiname: ")
        if not(filename in os.listdir(os.path.dirname(__file__))):
            print("Datei wurde nicht gefunden!\n")

    char_map, map, discovered, start, end = read_file(filename) # Einlesen der Eingabedatei

    # Finden des schnellsten Weges von A nach B, gibt letzten Wegpunkt des Weges zurück
    end_node = find_fastest_path(map, discovered, start, end)

    if end_node == None: # Es wurde kein Weg gefunden
        print("Es wurde kein Weg von A nach B gefunden!")
    else:
        output(char_map, end_node, filename) # Ausgeben der Lösung in der Konsole und Speichern in Textdatei
```